LIST

LONG ISLAND SINCLAIR TIMEX GROUP
INCORPORATING * NYTSE OF NEW YORK CITY
ISSUE: January 1990

* NEW YORK TIMEX SINCLAIR ENTHUSIASTS: NEXT MEETING Feb 11

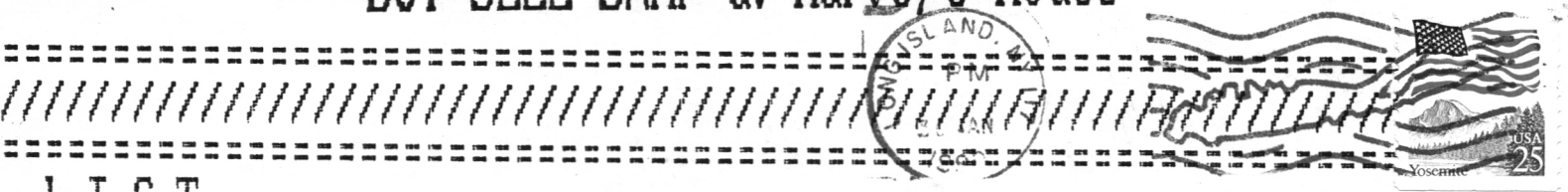
LIST MEMBERSHIP IS $15.00. LIBRARY TAPES ARE AVALIABLE, WRITE THE ADDRESS PRINTED BELOW.

NOTICE NOTICE NOTICE

SPE[illegible]UM UTILITY SPECTRUM UTILITY

USER GROUPS USER GROUPS

BUY SELL SWAP at Harvey's House



L.I.S.T.
5 PERI LANE
VALLEY STREAM, NY
11581

TO: *Don Lambert JAN/91*
3310 Clover Dr. S.W.
Cedar Rapids, IA
52404

FIRST CLASS MAIL

LIST OFFICERS

PRES.	HARVEY RAIT
TRES.	ROBERT MALLOY
REC.SEC.	STEVE KAYE
EDITOR.	FRED STERN
LIBR.	TOM SKAPINSKI

PLEASE SEND INQUIRIES TO:
LIST
MR. HARVEY RAIT
5 PERI LANE
VALLEY STREAM, N.Y. 11581
PLEASE SEND SUBMISSIONS TO:
LISTING
MR. FREDERIC STERN
214 ROBERTS ST.
HOLBROOK, N.Y. 11741

COMING EVENTS:

FEB. 03. 1990 UM PATERSON COLL. COMPUTER SHOW
FEB. 11. 1990 LIST MEET
FEB. 12. 1990 NYTSE MEETING
FEB. 17. 1990 RARITAN CENTER COMPUTER SHOW

NYTSE

NYTSE MEETS THE DAY AFTER THE LIST MEETING.
7:30 PM.
MISS KIMS
PARK AVENUE SOUTH
BETWEEN 21ST. AND 22ST.

MEETING MINUTES
JAN. 14. 1990

THE MEETING WAS CALLED TO ORDER AT 2:15 BY HARVEY.

WE WERE ALL SADDEN TO HEAR THAT HATS (HARRISBURG AREA TIMEX SINCLAIR USERS GROUP IS DISBANDING.
FRED WILL CONTACT HATS MEMBERS AND INVITE THEM TO JOIN LIST.

THE MARCH EDITION OF LISTING WILL LIST THE NAMES AND ADDRESSES OF ALL LIST MEMBERS. ANY MEMBER NOT WANTING HIS OR HER NAME AND ADDRESS PUBLISHED ARE ASKED TO SENT A POSTCARD TO THE LISTING EDITOR, FRED STERN 214 ROBERTS ST. HOLBROOK, N.Y. 11741.

WE RECEIVED WORD THAT; QL WORLD IS ENDING PUBLICATION. COMPUTER SHOPPER WILL NO LONGER HAVE THE TIMEX/SINCLAIR SURVIVAL COLUMN.

LIST MEMBER AND GOOD FRIEND YUGO DI GIOVIANNI IS IN THE HOSPITAL RECOVERING FROM A STROKE. CARDS, LETTERS, OR IF YOU CAN A VISIT WILL DEFINITELY HELP YUGOS RECOVERY.
CENTRAL GENERAL HOSPITAL
880 OLD COUNTRY ROAD
PLAINVIEW, N.Y. 11803
MR. YUGO DI GIOVIANNI
ROOM 154
VISITING: 12-2 AND 6-8

LIST MEMBER, LIBRARIAN AND GOOD FRIEND TOM SKAPINSKI SUFFERED A BROKEN ARM DURING THE CHRISTMAS HOLIDAYS.

WE WELCOME A NEW MEMBER CHRIS ZUBA. CHRIS IS A TEACHER FROM L.I. WHO USES A TS2068 FOR PLANNING AND LETTER WRITING.

THE SWAPMEET AND AUCTION

THE SWAPMEET WAS TRUELY A TIMEX SINCLAIR HACKERS DELIGHT.
BOOKS, SOFTWARE, HARDWARE AND COMPONENTS WERE AVAILABLE TO THE WISE SHOPPER.

THE AUCTION WAS AGAIN A HUGH SUCCESS. SOME OF THE BARGAINS;
MAGNAVOX RGB MONITOR $170
TS 2068 W/ (2) ZEBRA DISK DRIVES CONTROLLER, P/S, IN A CUSTOM MADE CABINET $150
EPSON PRINTER $70
TS1000 AND ZX81 W/KLICKIT KEYBOARDS, 16K RAMPACKS $15
TS2068 $40
BOOKS $1.00 EACH

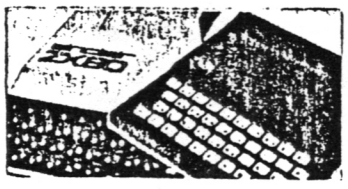
ZX-81,TS1000 TECHNICAL TIDBITS IS ON SALE THROUGH LIST FOR $4.00. PURCHASE YOUR COPY AT THE NEXT MEETING OR BY MAIL.

CLASSIFIEDS

THIS CLASSIFIED SECTION IS AVAILABLE TO ALL LIST MEMBERS FREE OF CHARGE.
THE ONLY RESTRICTION IS THAT IT IS TO BE USED ONLY FOR THE SEEKING, SELLING OR SWAPPING OF SINCLAIR, TIMEX OR MICROACE COMPUTER EQUIPMENT, PERIPHERALS AND SOFTWARE.
LISTING, LIST, AND ITS OFFICERS DO NOT ENDORSE, WARRANTY, OR GUARANTEE ANY OF THE ITEMS LISTED IN THIS CLASSIFIED SECTION

A FINAL WORD

MY NAME IS FRED STERN, AND I AM THE EDITOR OF THIS EDITON OF LISTING.
GET WELL WISHES GO OUT TO YUGO AND TOM FOR SPEEDY RECOVERIES.
I WISH TO INTRODUCE A NEW CONTRIBUTING WRITER, MR. WARREN FRICKE. ENCLOSED IS HIS ARTICLE ON MULTIPLICATION AND DIVISION FOR SIX GRADE LEVEL STUDENTS. THIS IS HIS FIRST OF HOPEFULLY MANY ARTICLES TO BE PUBLISHED IN LISTING.
THANKS ALSO GO TO JOHN PAZMINO FOR HIS CONTRIBUTION ON SPECTRUM PROGRAMS FROM CD.



NEWS

LIST exclusive bulletin

(Hey, are there any other kinds we got for you??)

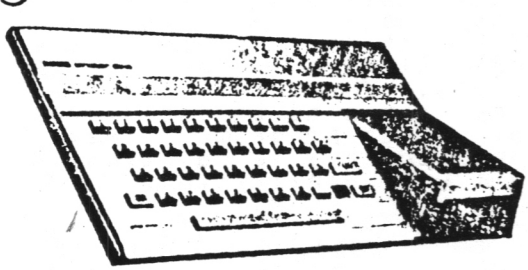
LOAD SPECTRUM PROGRAMS FROM CDs!!

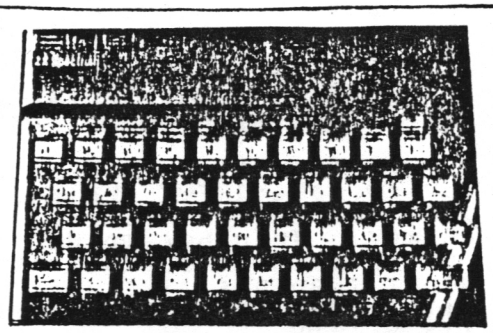
Yes, CodeMasters of the UK (Warwickshire, but plain UK will do, thank you) announce the first CD for the Spectrum (way to go!!). It has 30 of their games all packed onto an ordinary CD, which you load into your Spectrum with an ordinary CD player.

No, you do not need a CD-ROM discdrive; use the very same CD player you play audio CDs on. Complete kit consists of cable to connect CD player to Spectrum, tape with short MC program to allow the superfast bps rate of the CD, instructions for all 30 programs on the disc, and the very disc itself

Other UK firms are considering to follow CodeMasters' lead! Imagine, an adventure with thousands of levels! A war sim with millions of men in each army! Racing games with true crosscountry courses! Complete price is only £19.95!!

And you read it in LISTings! Naturally!





MULTIPLICATION, GRADE SIX

This tutorial demonstrates the simplicity with which a 64K computer can be programmed to provide a no-frill means of developing an unlimited number of multiplication problems for a math practice session of a six-grader. The program LISTing is short and can readily be converted to the basic language of other computers.

One of the biggest drawbacks of most software packages provided by the schools today for their students is that the programs are too elaborately written. Thru these programs the inanimate computers are expected to teach, but they cannot anticipate a child's needs, and are thus very inefficient as a teaching medium. As a consequence students can spend up to eighty percent of their time standing by.... watching crude computer animations, slow developing and fancy picture embellishments, repetitious displays of the credit and introductory screens, unclear menus, confusingly abbreviated instructions, and the like. Much of this rot is created by the programmers and producers of the software, who appear to be more concerned with demonstrating their programming ability than they are the education of the child. The relatively poor grades that students currently are scoring on standard tests, compared to the grades in the years before computers entered the classrooms, is principally due to this inefficient method of teaching. But, computers can be of help in providing a student with practice exercises to follow up on classroom instruction, by a teacher. H-34 is a program written for this purpose.

Routine H-34 does nothing more fancy than to develop two decimal numbers for multiplication on paper by the student who then enters his answer. The computer then checks it and indicates whether it is correct or not.

There are two subroutines in this program suitable for use in other programs. The one in lines 1000 to 1040 generates decimal and integral numbers in the range of 1 to 900+. They are developed as "strings" so that the individual digits can be manipulated. They are then converted to numbers for computation and display.

The subroutine in 500 to 680 reviews all INPUT and rejects anything that cannot be a valid answer. In other words, it monitors input so that nothing other than positive (or negative) numbers of 2 or more digits get by.

LONG DIVISION, GRADE SIX

Routine H-35 covers problems in long division and is similar in construction to H-34. The two routines could have been combined and supplied with a menu for choosing which operation to play, but I decided against this. Multiplication is usually taught several weeks before long division. They are not taught concurrently. Hence there is little to be gained by a combination.

It is very easy on the TS2068 to convert one similar program into another, as editing is simple on this computer. Starting with the LISTing of H-34, I altered the following lines....

5, 10, 20, 40, 80, 100, & 120

By comparing them you should be able to discern the particular action taken. But line 120 may need a further explanation. In H-34 we compared the student's response "try" with the computer's "ans", and allowed them to differ by no more than .03 in value. In division the difference is not a good criterion as all of the answers will be small. Therefore we used a ratio in the H-35 routine so that to score correctly, the difference between the two values divided by the computer's value agrees to within 6 significant figures. This requires that the student carry his work out to 6 figures, but it also limits his work to this extent.

H-35 has an added subroutine, lines 1500 to 1540, which generate the traditional division symbol which is not on most computers, including the Timex. This symbol, once generated, can be printed out simply by calling for CHR$ 146, or by the letter "A" in graphics. In this tutorial the graphics letter A was used and the symbol printed in its place. One of the many nice features of the Timex is its ability to mix graphics and text at any time and any where on the screen. Most other computers, operating in basic, cannot do this.

You can use this subroutine in other programs and can add up to 19 additional special symbols. More than 20 can be added but an explanation of the method will have to wait for an other day. The first two subroutines can also be incorporated into other programs. They may need to be altered slightly to suit your application.

Warren Fricke
273 Canton Street
Depew, NY 14043

H-34

```
   5 REM ** "I-16", 1-6-90, WF
  10 REM ** MULTIPLICATION,
            GRADE SIX

  20 RANDOMIZE
  30 CLS : LET F=0
  40 PRINT "You are expected to
be within    .03 of the correct
answer to       be marked correc
t."
  50 FOR I=1 TO 10
  60 GO SUB 1000: LET N1=VAL N$
  70 GO SUB 1000: LET N2=VAL N$
  80 LET ans=N1*N2
  90 PRINT '"QUESTION No. ";I
 100 PRINT "   ";N1;" X ";N2;" =
";
 110 INPUT T$: GO SUB 500
 115 LET try=VAL H$: PRINT try
 120 IF ABS (ans-try)>.03 THEN G
O TO 150
 130 PRINT "   Correct.  Press E
NTER."
 140 LET F=F+1: GO TO 160
 150 PRINT "   Wrong.  Answer is
 ";ans: PRINT "   Press ENTER."
 160 INPUT z$: NEXT I
 170 PRINT '"Your score was ";F;
" out of 10.        Press ENTER t
o play again."
 180 INPUT z$: GO TO 30
 190
 500 REM ** INPUT MONITOR
 510 LET H$=T$
 515 IF LEN T$<2 THEN GO TO 560
 520 IF T$(1)="+" OR T$(1)="-" T
HEN LET T$=T$(2 TO )
 530 GO TO 600
 560 FOR T=1 TO 120: NEXT T: GO
TO 110
 600 FOR T=1 TO LEN T$
 610 IF (T$(T)<"0" OR T$(T)>"9")
 AND T$(T)<>"." THEN GO TO 110
 620 NEXT T
 630 LET S=0
 640 FOR T=1 TO LEN T$
 650 IF T$(T)="." THEN LET S=S+1
 660 IF S>1 THEN GO TO 110
 670 NEXT T
 680 RETURN
 690
1000 REM ** NUMBER GENERATOR
1010 LET A$=STR$ (1+900*RND*RND)
+"000"
1020 LET W=2+INT (6*RND)
1030 LET N$=A$(1 TO W)
1040 RETURN
```

```
You are expected to be within
 .03 of the correct answer to
    be marked correct.

QUESTION No. 1
  197.9 X 11 = 2176.9
   Correct.  Press ENTER.

QUESTION No. 2
  3.3 X 354 = 1168.2
   Correct.  Press ENTER.

QUESTION No. 3
  144 X 114 = 16514
   Wrong.  Answer is 16416
   Press ENTER.
```

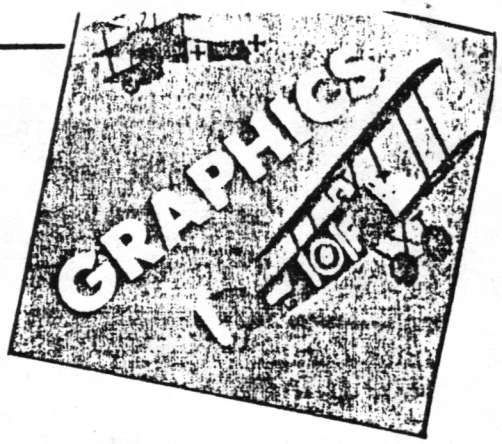
H-35

```
   5 REM ** "I-17", 1-6-90, WF
  10 REM ** LONG DIVISION,
            GRADE SIX

  20 RANDOMIZE : GO SUB 1500
  30 CLS : LET F=0
  40 PRINT "You are expected to
be correct  in the first 6 signi
ficant fig- ures to be marked co
rrect."
  50 FOR I=1 TO 10
  60 GO SUB 1000: LET N1=VAL N$
  70 GO SUB 1000: LET N2=VAL N$
  80 LET ans=N1/N2
  90 PRINT '"QUESTION No. ";I
 100 PRINT "   ";N1;" ÷ ";N2;" =
";
 110 INPUT T$: GO SUB 500
 115 LET try=VAL H$: PRINT try
 120 IF ABS (ans-try)/ans>1E-5 T
HEN GO TO 150
 130 PRINT "   Correct.  Press E
NTER."
 140 LET F=F+1: GO TO 160
 150 PRINT "   Wrong.  Answer is
 ";ans: PRINT "   Press ENTER."
 160 INPUT z$: NEXT I
 170 PRINT '"Your score was ";F;
" out of 10.        Press ENTER t
o play again."
 180 INPUT z$: GO TO 30
 190
 500 REM ** INPUT MONITOR
 510 LET H$=T$
 515 IF LEN T$<2 THEN GO TO 560
 520 IF T$(1)="+" OR T$(1)="-" T
HEN LET T$=T$(2 TO )
 530 GO TO 600
 560 FOR T=1 TO 120: NEXT T: GO
TO 110
 600 FOR T=1 TO LEN T$
 610 IF (T$(T)<"0" OR T$(T)>"9")
 AND T$(T)<>"." THEN GO TO 110
 620 NEXT T
 630 LET S=0
 640 FOR T=1 TO LEN T$
 650 IF T$(T)="." THEN LET S=S+1
 660 IF S>1 THEN GO TO 110
 670 NEXT T
 680 RETURN
 690
1000 REM ** NUMBER GENERATOR
1010 LET A$=STR$ (1+900*RND*RND)
+"000"
1020 LET W=2+INT (6*RND)
1030 LET N$=A$(1 TO W)
1040 RETURN
1050
1500 REM ** GENERATE ÷ SYMBOL
1510 FOR T=USR "a" TO USR "a"+7
1520 READ W: POKE T,W: NEXT T
1530 DATA 0,24,0,255,0,24,0,0
1540 RETURN
```

```
You are expected to be correct
in the first 6 significant fig-
ures to be marked correct.

QUESTION No. 1
  33 ÷ 77.8318 = 0.423991
   Correct.  Press ENTER.

QUESTION No. 2
  116.6 ÷ 12.9 = 9.03872
   Correct.  Press ENTER.

QUESTION No. 3
  209 ÷ 3.79436 = 55.182
   Wrong.  Answer is 55.081753
   Press ENTER.
```

Table 1.

Offset	No of Bytes	Parameter	Description	Range
0	1	NUMB	Number of sets of data.	1-255
1	2	ADDR	Start address of data.	
3	2	PX	X co-ord (+ VE LEFT)	
5	2	PY	Y co-ord (+ VE UP)	
7	2	PZ	Z co-ord (+ VE FORWARD)	
9	2	PHI	Angle about X axis	0-359°
11	2	THETA	Angle about Y axis	0-359°
13	2	PSI	Angle about Z axis	0-359°
15	2	DRAWS	Address of free memory after 3D data	
17	2	DRAWP	6*NUMB + DRAWS	
19	1	STFLG	Poke this with 0 the first time you use the 3D image.	0 or 255



3D ROTATOR

Mark Jones with a program which makes his 3D rotator published in the July 1983 issue up to eight times faster.

IN JULY OF LAST YEAR, I wrote a program for *Your Computer* called 3D Rotator. This program allowed the Basic programmer to manipulate simply defined 3D figures at machine-code speeds. A typical time was 0.5 seconds for a cube. Though this was extremely fast relative to Basic it was not fast enough for practical dynamic games. With this in mind I have speeded up the 3D routine by as much as eight times and made it more versatile. The speeds achieved now are as fast as those seen in commercial games such as 3D Tank Duel, with the advantage that they can be called from Basic.

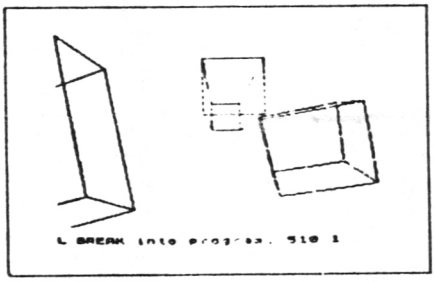
Data is stored as blocks of code at any memory location that you specify. The data should be followed by a blank area which = 12 * (number of sets of data). Therefore the total memory required for any one image = 19 * (number of sets of data). The data itself is made up of three 2's complement numbers and one 1 byte number. A 2's complement number is a 2 byte number where the negative form is 65536 – number, e.g., –5 = 65536 – 5 and +5 = 5: A 2 byte number is Poked into memory as described on page 173 of the Spectrum manual.

The numbers stored are as follows: x co-ordinate, y co-ordinate, z co-ordinate and the 1 byte number 0 to indicate a plot at x, y, z or 1 to draw a line from the last point plotted to x, y, z. I include a Basic program — program A — which will handle conversion of data into a suitable form for the machine-code program. The Basic program will also store the essential parameters such as the pointer to the data, number of sets of data etc., for that figure.

Producing the data for a simple 3D figure is relatively easy. I refer you for further information to my article 3D Rotator of July 1983 and Ian Angell's article BBC 3D Graphics in the February 1984 edition of *Your Computer*.

The 3D program allows the parameters for up to 16 3D images. These parameters are stored in fixed areas from 65032 onwards and in blocks of 20 bytes. Thus the start of the parameter area for figure 4 = 4*20 + 65032 — see table 1.

Each 3D image stored in memory should have a separate set of parameters though they



might share a common set of data. Draws need not point to a memory area after your 3D data but I find I keep track of my memory state better by doing so. For example, fig.0 and fig.1 might both be pyramids and so use the same data. Draws for fig.0 is as normal, after the data, but draws for fig.1 points at another section of free memory and its ADDR points to the fig.0 data.

If it has all seemed rather complicated so far, do not worry — it is really quite easy to use these 3D routines. Here is an example:

To set up fig.0 as a cube first of all work out your data and then store it in data statements in Basic program A. For a cube there are 16 sets of data so adjust line 15 accordingly. The data is going to be stored at 40000 onwards so first of all ensure this area is free from the Basic system with a

```
CLEAR 39999
```

and then adjust line 10 accordingly. Finally, this is going to be figure 0 so adjust line 5. Now run program A.

Once the program is complete it will give you a print-out of the next free memory available for data, 40316, and also the position of the parameters area, 65032. If you now wished to have another cube that could move independently of fig.0 then simply make line 10 read

```
LET ADDR = 40316
```

Line 5 should read

```
LET FIG = 1
```

and run the program again.

This is, of course, an example and actual addresses will depend on the number of sets of data you use. Program A as printed will set up fig.0 as a cube as in the above example.

Now to actually produce a 3D image on the screen there are a number of steps:

- Select the current figure by Poking 64976 with the required figure number 0-15.
- RAND USR 64234 actually converts your data to a list of plots and draws stored in the figure's associated free area of memory, pointed to by Draws.
- RAND USR 64692 produces the 3D image on the screen from the list of plots and draws.
- RAND USR 64679 deletes the last image drawn by the above routine.

Therefore using various sequences of these routines you can produce 3D images from within your Basic programs. As an added feature I have included a machine-code demonstration program which will put the current figure indicated by

```
PEEK 64976
```

through its paces. The number of steps, and so speed of this demonstration, can be altered by Poking 63501 with a number between 1 and 150.

The routine is called with

```
RAND USR 63500
```

Finally, to alter a given figures position on the screen, distance from you or angle simple alter the 2's complement numbers PX, PY, PZ, Phi, Theta and Psi in the parameters for that figure.

To run a demonstration of the routines from Basic load with program A and then Goto 5000. This sets up three cubes, shows off the machine-code demonstration and then leaves the three cubes to float around in space spinning.

If you remember the 3D Rotator routine in my last article you may be interested to compare it with these new routines. The routine to handle the 3D conversions is written more efficiently, uses 2 byte x, ,y,z co-ordinates and does not draw the figure straight away. This allows a number of figures to be produced

in memory but not drawn until needed.

This routine also handles calculations for lines partially off screen ensuring that the line eventually produced for the draw routine does not go off screen.

The routines to draw and delete figures use a draw routine specially written for this program which is extremely fast. The routine does not draw a line by plotting but by manipulating screen addresses and rotating a mask.

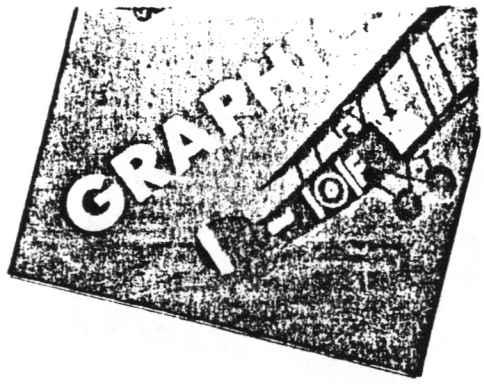
To use the plot/ routine for yourself use the following method:

- Set up an unused figure e.g., fig 15 by Poking both DrawS and DrawP with the same value, the address of an unused area of memory. Next Poke STFLG with 255.
- Store your plots and draws at this address in the following form:

P,x,y where P = 0 to plot at x,y
P = 1 to draw from the last point plotted to x,y
P = 255 to end the data
x is a normal x co-ordinate
Y has the range 0-191 where 0 is the bottom line of the edit area.
e.g. To draw a frame around the screen the data would be: 0,0,0, 1,255,0, 1,255,191, 1,0,191, 1,0,0, 255

Type program A into your computer and then Save it. Never try to run this program without the 3D machine-code routines in memory. For the 3D routines type in program B and then Save it. Now try running it. If an error is reported correct your error, resave the program and try again. Once the program has run successfully save the code as instructed in line 9999.

You now no longer need program B. You now have a machine-code program saved as code and a Basic program. Do not try and run the machine code without a valid set of parameters and data in memory. To see the various demonstrations load up with the Basic and then the code and then Goto 5000. Or to store an image type Run.



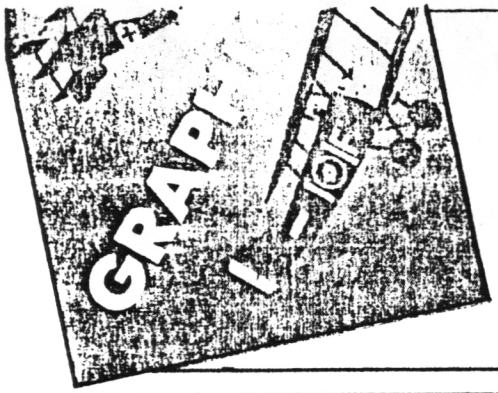
Program A.

```
   1 REM "Data Storage Program"
   2 CLS : CLEAR 39999
   4 REM
   5 LET fig=0: REM "Figure 0-15
"
   9 REM
  10 LET addr=40000: REM "data
storage address"
  14 REM
  15 LET No=16: REM "Number of
sets of data"
  16 GO SUB 17: STOP
  17 LET store=65032+fig*20: POK
E store,No: LET addr1=store+1: L
ET c=addr: GO SUB 500
  18 LET addr1=addr
  20 FOR f=1 TO No
  30 READ x,y,z,P
  40 LET c=x: GO SUB 500
  45 LET c=y: GO SUB 500
  50 LET c=z: GO SUB 500
  55 POKE addr1,P
  60 LET addr1=addr1+1
  75 NEXT f
  80 LET spare=12*(No+1)
  82 LET free=spare+addr1
  85 LET c=addr1: LET addr1=stor
e+15: GO SUB 500: LET c=spare/2+
c: GO SUB 500: POKE addr1,0
  95 PRINT "Data stored for figu
re ";fig
 100 PRINT "data at ";addr
 105 PRINT "next free memory at
";free
 110 PRINT "Parameters at ";stor
e;" to ";store+19
 120 RETURN
 500 IF c<0 THEN  LET c=65536+c
 505 POKE addr1+1,INT (c/256)
 510 POKE addr1,INT (256*(c/256-
INT (c/256)))
 515 LET addr1=addr1+2
 520 RETURN
 999 REM cube data
1000 DATA 20,-20,20,0,20,-20,-20
,1,-20,-20,-20,1,-20,-20,20,1,20
,-20,20,1,20,20,20,1,20,20,-20,1
,-20,20,-20,1,-20,20,20,1,20,20,
20,1,20,-20,-20,0,20,20,-20,1,-2
0,-20,-20,0,-20,20,-20,1,-20,-20
,20,0,-20,20,20,1
4997 REM
4998 REM  Demonstration
4999 REM
5000 RESTORE : LET fig=0: LET ad
dr=40000: LET No=16: GO SUB 17
5010 RESTORE : LET fig=1: LET ad
dr=free: GO SUB 17: RESTORE : LE
T fig=2: LET addr=free: GO SUB 1
7
5011 POKE 63501,10:  CLS : POKE
64976,1: RANDOMIZE USR 63500: PO
KE 64976,0: RANDOMIZE USR 63500:
 POKE 64976,2: RANDOMIZE USR 635
00
5012 FOR f=5  TO 95 STEP 10: POK
E 63501,f: RANDOMIZE USR 63500:
NEXT f
5015 FOR f=65081 TO 65086: POKE
f,0: NEXT f
5016 POKE 63501,10
5019 POKE 65075,40: POKE 65079,2
00: POKE 65036,0: POKE 65056,0:
POKE 65038,0: POKE 65037,0: POKE
 65058,0: POKE 65057,0: POKE 650
39,150: POKEEE 65040,0: POKE 65059
,20: POKE 65060,0
5020 FOR f=170 TO -170 STEP -30:
 LET c=f: LET addr1=65035: GO SU
B 500: LET addr1=65077: GO SUB 5
00: LET c=-f: LET addr1=65055: G
O SUB 500: LET g=ABS (f): POKE 6
5041,g: POKE 65063,g
5025 FOR h=0 TO 2: GO SUB 6000:
NEXT h
5040 NEXT f
5041 RANDOMIZE USR 63500: GO TO
5015
6000 POKE 64976,h: RANDOMIZE USR
 64234: RANDOMIZE USR 64679: RAN
DOMIZE USR 64692: RETURN
```

Program B.

```
  10 DATA "3e0afefa38023efa06004
f21fe00c5e5dde1e5cdd7fae5d5c5edb
0dde5e122d8fd22defd22dcfd22dafd1
1ff00ebed52cb3ccb1dcb3ccb1d22d4f
d22d6fdc1e1d1edb0cdeafacda7fccdb
4fce1c1ed427ccb7f2008fe0020b579b
d38b1fbc9c1c1c9c5ed"
  11 DATA 16344
  20 DATA "4beffdc53ebf9038f2cdb
0224704 3e010f10fde5d9e1d908e1c1e
d43effd1101017894300 4ed4416ff477
9953004ed441eff4fb8300969d5d9d1d
9af5f180bb128666841d5d9d1d916006
0781f853803bc3807944fd94b4218044
fd5d9c1cb402829cb78"
  21 DATA 11938
  30 DATA "20153e07a4280325181d2
57dd6206f38167cc60867181024 3e07a
4200a7dc6206f38047cd6086708cb412
80ecb7920060f30072c18040730012d4
700b6777808d97910a5c9fdcb45862a0
5fe110001ed52ed5bd8fd19cb7cc43df
afd7e452205fe22e5fd"
  31 DATA 91400
  40 DATA "2a01fe2929ed5bd4fd19f
5cd6af9118000cda1f92201fe2a03fe2
929ed5bd6fd19f1fd7745cd6af911570
0cda1f92203fec9cb7cc43dfaeb2ae5f
de5ed52e13801eb444daf08af086f67c
b38cb19200508b9280b08300319ce00e
b29eb18eb08300319ce"
  41 DATA 12206
  50 DATA "00456c67c9ebfdcb45462
80619fdcb4586c9ed52c9e57cb52814c
df9f922f3fde1cd05fa22f1fd21f1fd0
10400c9e13e8032f1fd21f1fd0101010c
92adcfdcdb1f911f5fdedb02adefdcdb
1f911f9fdedb02adafdcdb1f911fdfde
db0c9015a0009016801"
  51 DATA 14865
```

(Program B continued on next page)



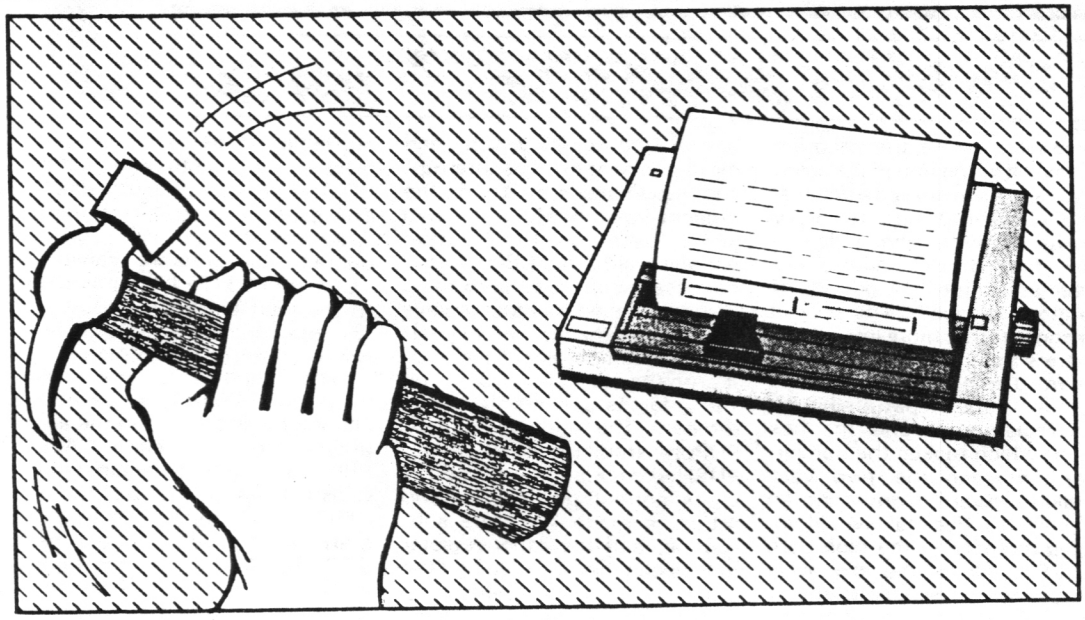
(program B continued from previous page)

```
  60 DATA "ed42300109af3c01b400e
d4230023d09011afd0956eb6fc9fdcb4
7462808dd6e03dd7e021806dd6e01dd7
e00260022e5fdfd7745ebcd6af9fdcb4
546c8af575febed52fd3445c9ddcb007
ec0c5cd18fafd3447d1e5cd18fad1fdc
b47462803l9afc9ebed"
  61 DATA 12663
  70 DATA "52afc9dd21f5fded5b01f
eed4b05fefdcb47c6cd47fa2013ed5b0
1fe2201feed4b05fecd47fa2205fe444
ddd21fdfded5b03fefdcb47c6cd47fa2
011ed5b03fe2203feed4b05fecd47fa2
205feed4b03fedd21f9fded5b01fefdc
b47c6cd47fac0ed5b01"
  71 DATA 14664
  80 DATA "fe2201feed4b03fecd47f
a2203fec93ad0fd2108fe11d1fd01140
0a7c8093d20fcc9cdd7faedb03ae4fda
720083eff2b772ae0fd77af3207fecdd
7f9ed5be2fd3ad1fd472ad2fdc5d55e2
356ed5301fe235e2356ed5303fe235e2
356ed5305fe23e5cd67"
  81 DATA 13798
  90 DATA "facd19f9e17e23d1e5f5d
53a07fe4f2a01feed5b03feafb4b2200
73ebfbb3e0030023e013207feb1201b2
2ebfded53edfdc1f102037d02037b020
35950e1c110a23eff12c97ca7280b3ae
cfda72805ac17d29bfc7aa7280d3aeef
da72807aa17d29bfc18"
  91 DATA 12207
 100 DATA "0d7bfec038083aedfdfec
0d29bfce5d5e52aedfda7ed5222e7fdd
12aebfda7ed5222e9fdebed5b03fecde
9fbe5d52aebfded5bedfdcde9fb6322e
9fdd1e1c1ed43edfdc1ed43ebfdc1af0
2033ae9fd02033aeafd0203c35efbd5c
b7c2806cd47fc210000"
 101 DATA 16381
 110 DATA "7ca7280c11ff00a7ed52c
d47fc21ff00e52ae9fded5be7fded53e
9fd22e7fdd1e1d5cb7c2806cd47fc210
0007ca72005 7dfec0380c11bf00a7ed5
2cd47fc21bf00e52ae9fded5be7fded5
3e9fd22e7fdd1e1c9fdcb4586ed5be7f
dcd8dfc6568ed5be9fd"
 111 DATA 15420
 120 DATA "cd63fcc1d1cda1f9e5c5c
9ebcb7cc487fceb7ab3281501000 0a7e
d5238030318f91929ed523801036960c
9210000c9d5cd3dfad1c9cb7cc487fce
bcb7cc487fcc373f922ebfded53edfdd
1f1c369fbcdf6fce50e2f06a62ae0fd1
816cdf6fce50e0006b6"
 121 DATA 14795
 130 DATA "2ae2fded5be0fd22e0fde
d53e2fded430ff9d9e5d97e23feff281
24e234623e51f3804ed43effdcd6ef8e
118e8d9e1d9d121e0fd010400edb0c9c
dd7fa010f000911e0fd010500e5edb03
ae4fda7200c2b3eff772ae0fd772ae2f
d77e1c900040080d1116"
 131 DATA 13804
 140 DATA "1a1f23282c3035393d424
64a4f53575b5f64686c7074787c7f838
78b8f92969a9da1a4a7abaeb1b5b8bbb
ec1c4c6c9cccfd1d4d6d9dbdddfe2e4e
6e8e9ebedeef0f2f3f4f6f7f8f9fafbf
cfcfdfefeffffffffffffffffffffff
efefdfcfcfbfaf9f8f7"
 141 DATA 18475
 150 DATA "f6f4f3f2f0eeedebe9e8e
6e4e2dfdddbd9d6d4d1cfccc9c6c4c1b
ebbb8b5b1aeaba7a4a19d9a96928f8b8
783807c7874706c68645f5b57534f4a4
6423d3935302c28231f1a16110d080400
00000"
 151 DATA 10715
4990 CLEAR 59999
5000 LET c=0: LET f=63500
5010 FOR h=1 TO 15: READ a$
5020 FOR s=1 TO LEN a$ STEP 2
5030 LET a=CODE a$(s): LET b=COD
E a$(s+1)
5050 IF a>96 THEN  LET a=a-39
5060 IF b>96 THEN  LET b=b-39
5070 LET a=a-48
5080 LET b=b-48
5090 POKE f,a*16+b
5091 LET c=c+PEEK f
5100 LET f=f+1
5110 NEXT s
5111 PRINT h
5120 READ tot: IF tot<>c THEN  P
RINT "Error at line ";h*10: STOP
5125  LET c=0
5130 NEXT h
5131 REM
9999 CLS : BEEP .2,0: PRINT "COD
E STORED CORRECTLY": PRINT "SAVE
 CODE 63500,1478"
```

Printer Ploys

Charles A Barron gets to grips with using his printer.

From ZX Computing



So Uncle Clive has killed off the ZX Printer – one of the most useful and impressive little toys he ever gave us. But then he gave us Interface I and who can resist plugging in a real live full-width printer to its little RS232 port?

But now that we all have a real printer, can we make best use of it? The first essential is some software to make the thing work properly – something like *Tasword 2*, perhaps, which will format our print-outs with neatly justified right-hand margins. But that's just a start. Do your documents involve typing the same name or formula over and over again? Well, if your software allows you to seek and replace a word, you can save yourself a great deal of finger-tapping. For example, I spend most of my hours at my Spectrum pretending it's a word-processor in order to write plays. Now the one thing you can be sure of in writing a play is that you are going to have a lot of repetitive typing: every time a character speaks you need his name in full. A page of quick-fire dialogue may use a character's name 20 or 30 times, especially when you consider that you not only begin every speech with the speaker's name, you also have the characters constantly addressing each other by name: You've seen the kind of thing:

MURGATROYD: Daphne!
DAPHNE: Murgatroyd! It's you!
MURGATROYD: As you say Daphne, it is I.
DAPHNE: At last, Murgatroyd.

Four *Murgatroyds* and four *Daphnes* in just four lines of deathless dialogue. Most writers cheat, when writing their draft versions of the text, and use abbreviations for the names. Ever noticed how characters' names always begin with different letters of the alphabet? That way we can just use the initial letter for identification in the early scripts. (We're way ahead of one William Shakespeare here; Macbeth, Malcolm, Macduff – all in one play. He must have been fonder of quill-scratching than I am of keyboard-tapping.)

Once the rough draft has been cleaned up and polished, you just have to use the *find-and-replace* function to find all your character-identification initials and replace them with the names in full. So, the draft looks like this:

M: D!
D: M! It's you!
M: As you say, D. It is I.
D: At last, M.

And the filled out version looks like the quotation above. A saving of 56 key-taps in four lines!

Once the play, or whatever, is complete, you'll want to give it a finished look. Page numbers. Maybe even page headers? (That is where the title of the piece appears at the top of every page.) An unconscionable amount of typing for just a little fancy decoration. And none of the software packages for the Spectrum that I've come across allows automatic page numbering. That is the only really desirable function of those £5000 word processors that is missing on Uncle Clive's little wonder. (Though by the time you've added Interface and the printer and a couple of microdrive cartridges to store your prose on, it is beginning to cost about as much as the £5000 job!).

Here is a little program that will give you automatic page numbering and automatic page headers at the cost of typing it not only once. It will only work if your printer/interface/software combination allows you to program in a form feed instruction: that moves the paper through the printer one full fanfold. You should also set them to give you automatic skip over the perforations, if that is possible.

Run off your document, keeping the perforations intact; tear off the last sheet and then feed in page one again, setting the top of the page carefully to where you would want the page header to come. Set the program running and the printer will run your printed play or White Paper through again, pausing only once every page to add the titles and page numbers.

In the program listing, which should work for the common types of dot matrix printers, you will have to set the baud rate to suit your own set-up, and put enough spaces into your title in line 20 to bring it to the desired position on the page. If your printer can be programmed with TAB settings, then you can use these instead of the row of spaces. The program begins by asking you how many pages your opus runs to. If you are too tired after composing the thing to face counting them, you can always lie to your computer and pretend to have written 2000 pages – it will give up the program of its own accord when it runs out of paper. (But it may never trust you to tell the truth again!).

CHR$ 12 is simply the Form Feed control code; your machine may need a different code, though that is unlikely. Refer, as they say, to the manual and adjust the program accordingly. 'As The Bat At Noon' is what we call in the trade a variable; please don't name all your documents after my play.

PROGRAM LISTING

```
3     "No of pages?";n
5     FORMAT "b";4800: OPEN #3;"b"
10    FOR a = 1 TO n
20    LPRINT "    As The Bat At Noon    ";a
30    LPRINT CHR$ 12
40    NEXT a
50    CLOSE #3
```

Machine Code Trace

Coventry's Carol Brooksbank wrote this utility to find bugs in her programs and she thought she'd share it with us.

From 2x computing 1986

I don't know about you, but I don't think that I have ever written a machine code program which ran perfectly first time. You know the feeling. Eagerly, you type in your latest masterpiece, enter RANDOMIZE USR something-or-other and **CRASH!** There you sit, with a frozen keyboard and only a blank screen or a pretty psychedelic pattern to look at. You have no idea whether there is a fault in the logic of your program, whether you have made a typing mistake or miscounted a displacement, and you don't know where to start looking for the trouble, because you do not know how far into the program the crash occurred. Well, help is at hand.

This machine code program will give you a hex trace at the right of the screen, as your program runs. It is only a partial trace, as it does not show the address of every instructions as it is executed — if it did, the display would change so fast that you would never be able to read it — but every 1/50 second it gives you the address that the program has reached. This is enough to let you keep an eye on the progress of your program, and to see where things start to go wrong. For instance, if the crash is caused by the program getting into an endless loop, you will see the same sequence of bytes repeated over and over again after the crash happens. If you left out a return instruction, so that the program starts running through the empty bytes above your program, that too will be obvious. But remember that the trouble is not always at the point where the crash happens. A wrong displacement instruction may be some way away from the point to which it directs the program. You will still have to think for yourself to decide why the program runs as it does.

Why is the display in hex? Two reasons. The first is purely personal. I wrote the program for myself in the first place, and I always work in hex, so a decimal display would not be very helpful. (One of these days I shall find myself asking the greengrocer for "0A pounds of potatoes, please".) The second reason is rather more important. There is a direct relationship between the binary form of a number — the bit pattern held in the registers — and the hex form, which makes the conversion between the two very straightforward. Converting an address to decimal would involve multiplying the high byte by 256, adding the low byte, then isolating the 5 digits one by one for printing, all of which would make the routine much more complicated. Since the trace routine is in the form of an interrupt subroutine, it is desirable that it should be as short and simple as possible.

The routine makes use of the fact that, whenever the Spectrum performs a subroutine, the return address is pushed onto the stack. On an interrupt subroutine, the return address is the program counter, the point reached in the main program. If we can retrieve this address from the stack and display it, we have a trace. Obviously, there are a lot of instructions performed in between the interrupts which are not displayed, but this is usually enough to let you see where a program takes a wrong turning. So, if your machine code program crashes, load in this routine with your own program — I am assuming that you always take the precaution of taping your programs before running them, just in case — enter RANDOMIZE USR 65271 (48K), 32503 (16K), run your program again, and all should be revealed.

Details

The program is explained by the notes in the listing, but there are one or two details which need more explanation. The interrupt subroutine starts by saving the present value of HL in the two spare bytes in the system variables area at 5CB0. This is necessary because the existing values of registers must always be saved at the start of an interrupt subroutine, and if we push it onto the stack, it will cover up the address we are trying to retrieve. The address is then POPped from the stack in HL, PUSHed back again so that it is in its correct place when the return is made from the subroutine, and the other register values can then be saved on the stack. The other spare byte among the system variables, 5C81, is used as an interrupt counter. If this has reached 22d, the printing position is set to the top of the screen and the counter reset to 0. Otherwise, the routine jumps forward to print the address.

The print subroutine starts with the instruction AND 0F, which has the effext of resetting bits 4-7 of the A register, leaving bits 0-3 unchanged, isolating the number we wish to print. PRINT must be called, therefore, with the number to be printed in bits 0-3 of A. If the number to be printed is the "left hand" digit of the two in the A register, the instruction RRA is performed 4 times, to move it to the "right-hand" position, but the print subroutine is called directly when the "right-hand" digit it to be printed. When PRINT is called, the DE register holds the first byte of the screen position for the digit, and at the end of the PRINT subroutine, DE is restored to that position.

Since there are only 16 digits which we shall need to print, 0-9 and A-F, a table is set up, starting at FED7 (7ED7 16K), which holds the start addresses of the bit patterns of those digits in the ROM character table. Doubling the value of the number to print and adding it to the address of our table, points to the correct place in the table to retrieve the ROM address for that character. The digit can then be printed. After the 4 digits have been printed, the program variable SCRP at FF13 (7F13) is pointed to the next screen row down, and the program exits via the normal interrupt subroutine.

The listing is for the 48K machine. 16K folk should change the initial "F" in the addresses to "7", each CALL PRINT instruction should read CDB97E, and the bytes at 7EBD, which point to HL to the start of the table should be 21D77E. At START, the high byte of the interrupt vector address should be 28, giving the bytes 3E28. The interrupt vector address is not required at 7EFF, so the four bytes between 7EFD and 7F01 may be changed to NOP if you wish, though if they are left as they are the program will simply ignore them.

Saving

To SAVE the routine on tape:

SAVE "m/c trace" CODE 65116, 184 (48K)
SAVE "m/c trace" CODE 32348, 184 (16K)

To START the trace:

RANDOMIZE USR 65271 (48K)
RANDOMIZE USR 32503 (16K)

To STOP the trace:

RANDOMIZE USR 65292 (48K)
RANDOMIZE USR 32524 (16K)

Finally, remember that the trace will not work if the interrupts are disabled. You must change your DI and EI instructions to NOP while using the trace, and restore them when you have corrected your problems. ▶

MACHINE CODE TRACE PROGRAM LISTING

ADD.	M/CODE	LABEL	ASSEMBLY	NOTES
FE5C	22B05C	INT S/R	LD(5CB0),HL	Save present value of HL
FE5F	E1		POP HL	Fetch program counter
FE60	E5		PUSH HL	Save it again
FE61	F5		PUSH AF	Save
FE62	C5		PUSH BC	all
FE63	D5		PUSH DE	registers
FE64	3A815C		LD A,(5C81)	Fetch program counter
FE67	3C		INC A	update and
FE68	32815C		LD(5C81),A	store it again
FE6B	FE16		CP 16	Has counter reached 22d?
FE6D	200B		JRNZ CONT	Jump forward if not
FE6F	111C40		LD DE,401C	Set variable to
FE72	ED5313FF		LD(SCRP),DE	first screen position
FE76	AF		XOR A	Set counter
FE77	32815C		LD(5C81),A	to 0
FE7A	ED5B13FF	CONT	LD DE,(SCRP)	Fetch current screen position
FE7E	7C		LD A,H	Fetch first two digits
FE7F	1F		RRA	First digit
FE80	1F		RRA	to bits
FE81	1F		RRA	0 - 3
FE82	1F		RRA	of A register
FE83	CDB9FE		CALL PRINT	Print first digit
FE86	13		INC DE	Point to next screen position
FE87	7C		LD A,H	Fetch second digit
FE88	CDB9FE		CALL PRINT	Print second digit
FE8B	13		INC DE	Next screen position
FE8C	7D		LD A,L	Fetch last two digits
FE8D	1F		RRA	Third digit
FE8E	1F		RRA	to bits
FE8F	1F		RRA	0 - 3
FE90	1F		RRA	of A register
FE91	CDB9FE		CALL PRINT	Print third digit
FE94	13		INC DE	Next screen position
FE95	7D		LD A,L	Fetch last digit
FE96	CDB9FE		CALL PRINT	Print last digit
FE99	2A13FF		LD HL,(SCRP)	Fetch current screen position
FE9C	CB1C		RR H	Point HL
FE9E	CB1C		RR H	to next
FEA0	CB1C		RR H	screen row
FEA2	012000		LD BC,0020	down
FEA5	ED4A		ADD HL,BC	
FEA7	CB14		RL H	
FEA9	CB14		RL H	
FEAB	CB14		RL H	
FEAD	2213FF		LD(SCRP),HL	Save new screen position
FEB0	D1		POP DE	Restore
FEB1	C1		POP BC	all
FEB2	F1		POP AF	registers
FEB3	2AB05C		LD HL,(5CB0)	Restore HL
FEB6	C33800		JP0038	Exit via normal interrupt s/r
FEB9	E60F	PRINT	AND 0F	Isolate number to print
FEBB	87		ADD A,A	Double it
FEBC	E5		PUSH HL	Save program counter
FEBD	21D7FE		LD HL,FED7	Start of table to HL
FEC0	0600		LD B,00	Displacement to
FEC2	4F		LD C,A	BC
FEC3	09		ADD HL,BC	Add to start of table
FEC4	46		LD B,(HL)	Fetch ROM
FEC5	23		INC HL	character table
FEC6	4E		LD C,(HL)	address for digit
FEC7	C5		PUSH BC	and transfer to
FEC8	E1		POP HL	HL
FEC9	0608		LD B,08	Counter of bytes to print
FECB	7E	RPT	LD A,(HL)	Fetch byte to print
FECC	12		LD(DE),A	Print it
FECD	23		INC HL	Point to next character byte
FECE	14		INC D	Point to next screen byte
FECF	10FA		DJNZ RPT	Jump back unless 8 bytes printed
FED1	7A		LD A,D	Restore DE
FED2	D608		SUB 08	to screen position

```
FED4 57                     LD D,A          for digit 1
FED5 E1                     POP HL          Fetch program counter
FED6 C9                     RET             Exit subroutine
FED7 3D80       TABLE       DEFB
FED9 3D88                   DEFB
FEDB 3D90                   DEFB
FEDD 3D98                   DEFB
FEDF 3DA0                   DEFB
FEE1 3DA8                   DEFB
FEE3 3DB0                   DEFB
FEE5 3DB8                   DEFB
FEE7 3DC0                   DEFB
FEE9 3DC8                   DEFB
FEEB 3E08                   DEFB
FEED 3E10                   DEFB
FEEF 3E18                   DEFB
FEF1 3E20                   DEFB
FEF3 3E28                   DEFB
FEF5 3E30                   DEFB
FEF7 211C40     START       LD HL,401C      Store first
FEFA 2213FF                 LD(SCRP),HL     screen position
FEFD 1802                   JR PASS         By-pass
FEFF 5CFE                   DEFB            interrupt vector address
FF01 AF         PASS        XOR A           Set interrupt counter
FF02 32815C                 LD(5C81),A      to 0
FF05 3EFE                   LD A,FE         High byte of interrupt
FF07 ED47                   LD I,A          vector address to I register
FF09 ED5E                   IM2             Select interrupt mode 2
FF0B C9                     RET             Return
FF0C ED56       STOP        IM1             Select normal interrupt mode
FF0E 3E3F                   LD A,3F         Normal interrupt value
FF10 ED47                   LD I,A          to I register
FF12 C9                     RET             Return
FF13 0000       SCRP        DEFB            Program variable
```

```
15F8
15E2
10B4
1620
15EC
15E8
15FF
10AF
15F8
15E8
15E1
10AF
15F8
15E8
15E1
10AF
15F8
0E5C
15F8
15E8
15E1
10AF
```

SCREEN DUMP OF MACHINE CODE TRACE DISPLAY